A Unix Person's Guide to PowerShell

by Matt Penny

PowerShell.org

Table of Contents

ReadMe	0
About this Book	1
Introduction to PowerShell for Unix People	2
Commands Summary	3
Command Detail - A	4
Command Detail - B	5
Command Detail - C	6
Command Detail - D	7
Command Detail - E	8
Command Detail - F	9
Command Detail - G	10
Command Detail - H	11
Command Detail - I	12
Command Detail - J	13
Command Detail - K	14
Command Detail - L	15
Command Detail - M	16
Command Detail - N	17
Command Detail - O	18
Command Detail - P	19
Command Detail - Q	20
Command Detail - R	21
Command Detail - S	22
Command Detail - T	23
Command Detail - U	24
Command Detail - V	25
Command Detail - W	26
Command Detail - X	27
Command Detail - Y	28
Command Detail - Z	29

Command Detail - Non-alphabetical 30
To-do 31

This e-book is intended as a 'Quick Start' guide to PowerShell for people who already know Bash or one of the other Unix shells.

The book has 3 elements:

- an introductory chapter which covers some PowerShell concepts
- a summary list of PowerShell equivalents of Unix commands in one e-book chapter
- a detailed discussion of Powershell equivalents of Unix commands, organised in the alphabetical order of the unix command

About

Principal author: Matt Penny

This e-book is intended as a 'Quick Start' guide to PowerShell for people who already know Bash or one of the other Unix shells.

The book has 3 elements:

- an introductory chapter which covers some PowerShell concepts
- a summary list of PowerShell equivalents of Unix commands in one e-book chapter
- a detailed discussion of Powershell equivalents of Unix commands, organised in the alphabetical order of the unix command

---

Was this book helpful? The author(s) kindly ask(s) that you make a tax-deductible (in the US; check your laws if you live elsewhere) donation of any amount to The DevOps Collective to support their ongoing work.

Check for Updates! Our ebooks are often updated with new and corrected content. We make them available in three ways:

- Our main, authoritative GitHub organization, with a repo for each book. Visit https://github.com/devops-collective-inc/
- Our GitBook page, where you can browse books online, or download as PDF, EPUB, or MOBI. Using the online reader, you can link to specific chapters. Visit https://www.gitbook.com/@devopscollective
- On LeanPub, where you can download as PDF, EPUB, or MOBI (login required), and "purchase" the books to make a donation to DevOps Collective. You can also choose to be notified of updates. Visit https://leanpub.com/u/devopscollective

GitBook and LeanPub have slightly different PDF formatting output, so you can choose the one you prefer. LeanPub can also notify you when we push updates. Our main GitHub repo is authoritative; repositories on other sites are usually just mirrors used for the publishing

process. GitBook will usually contain our latest version, including not-yet-finished bits; LeanPub always contains the most recent "public release" of any book.

Introduction to PowerShell for Unix people

The point of this section is to outline a few areas which I think *nix people should pay particular attention to when learning Powershell.

Resources for learning PowerShell

A *full* introduction to PowerShell is beyond the scope of this e-book. My recommendations for an end-to-end view of PowerShell are:

- Learn Windows PowerShell in a Month of Lunches - Written by powershell.org's Don Jones and Jeffery Hicks, I would guess that this is the book that most people have used to learn Powershell. It's 'the Llama book' of Powershell.
- Microsoft Virtual Academy's 'Getting Started with PowerShell' and 'Advanced Tools & Scripting with PowerShell' Jump Start courses - these are recordings of day long webcasts, and are both free.

unix-like aliases

PowerShell is a friendly environment for Unix people to work in. Many of the concepts are similar, and the PowerShell team have built in a number of Powershell aliases that look like unix commands. So, you can, for example type:

```
ls
```

....and get this:

```
    Directory: C:\temp
Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---        22/02/2015     16:51      25773 all_the_details.md
-a---        20/02/2015     07:31       3390 commands-summary.md
```

These can be quite useful when you're switching between shells, although I found that it can be irritating when the 'muscle-memory' kicks in and you find yourself typing `ls -ltr` in PowerShell and get an error. The 'ls' is just an alias for the PowerShell `get-childitem` and the Powershell command doesn't understand `-ltr` [1].

the pipeline

The PowerShell pipeline is much the same as the Bash shell pipeline. The output of one command is piped to another one with the ' `|` ' symbol.

The big difference between piping in the two shells is that in the unix shells you are piping *text*, whereas in PowerShell you are piping *objects*.

This sounds like it's going to be a big deal, but it's not really.

In practice, if you wanted to get a list of process names, in bash you might do this:

```
ps -ef | cut -c 49-70
```

...whereas In PowerShell you would do this:

```
get-process | select ProcessName
```

In Bash you are working with characters, or tab-delimited fields. In PowerShell you work with field names, which are known as 'properties'.

get-help, get-command, get-member

get-member

When you run a PowerShell command, such as `get-history` only a subset of the `get-history` output is returned to the screen.

In the case of `get-history`, by default two properties are shown - 'Id' and 'Commandline'...

```
$ get-history

  Id CommandLine
  -- -----------
   1 dir -recurse c:\temp
```

...but get-history has 4 other properties which you might or might not be interested in:

```
$ get-history | select *

Id                 : 1
CommandLine        : dir -recurse c:\temp
ExecutionStatus    : Completed
StartExecutionTime : 06/05/2015 13:46:56
EndExecutionTime   : 06/05/2015 13:47:07
```

The disparity between what is shown and what is available is even greater for more complex entities like 'process'. By default `get-process` shows 8 columns, but there are actually over 50 properties (as well as 20 or so methods) available.

The full range of what you can return from a PowerShell command is given by the `get-member` command[2].

To run `get-member`, you pipe the output of the command you're interested in to it, for example:

```
get-process | get-member
```

....or, more typically:

```
get-process | gm
```

`get-member` is one of the 'trinity' of 'help'-ful commands:

- get-member
- get-help
- get-command

get-help

`get-help` is similar to the Unix `man` [3].

So if you type `get-help get-process`, you'll get this:

```
NAME
    Get-Process

SYNOPSIS
    Gets the processes that are running on the local computer or a remote computer.


SYNTAX
    Get-Process [[-Name] <String[]>] [-ComputerName <String[]>] [-FileVersionInfo] [-Modu

    Get-Process [-ComputerName <String[]>] [-FileVersionInfo] [-Module] -Id <Int32[]> [<C

    Get-Process [-ComputerName <String[]>] [-FileVersionInfo] [-Module] -InputObject <Pro


DESCRIPTION
    The Get-Process cmdlet gets the processes on a local or remote computer.

    Without parameters, Get-Process gets all of the processes on the local computer. You
    process by process name or process ID (PID) or pass a process object through the pipe

    By default, Get-Process returns a process object that has detailed information about
    methods that let you start and stop the process. You can also use the parameters of G
    version information for the program that runs in the process and to get the modules t


RELATED LINKS
    Online Version: http://go.microsoft.com/fwlink/?LinkID=113324
    Debug-Process
    Get-Process
    Start-Process
    Stop-Process
    Wait-Process

REMARKS
    To see the examples, type: "get-help Get-Process -examples".
    For more information, type: "get-help Get-Process -detailed".
    For technical information, type: "get-help Get-Process -full".
    For online help, type: "get-help Get-Process -online"
```

There are a couple of wrinkles which actually make the PowerShell 'help' even more *help*-ful.

- you get basic help by typing `get-help`, more help by typing `get-help -full` and...probably the best bit as far as I'm concerned...you can cut to the chase by typing `get-help -examples`

- there are lots of ' `about_` ' pages. These cover concepts, new features (in for example `about_Windows_Powershell_5.0`) and subjects which dont just relate to one particular command. You can see a full list of the 'about' topics by typing `get-help about`
- get-help works like `man -k` or `apropos` . If you're not sure of the command you want to see help on, just type `help process` and you'll see a list of all the help topics that talk about processes. If there was only one it would just show you that topic
- *Comment-based help*. When you write your own commands you can (and should!) use the comment-based help functionality. You follow a loose template for writing a comment header block, and then this becomes part of the get-help subsystem. It's good.

get-command

If you don't want to go through the help system, and you're not sure what command you need, you can use `get-command` .

I use this most often with wild-cards either to explore what's available or to check on spelling.

For example, I tend to need to look up the spelling of `ConvertTo-Csv` on a fairly regular basis. PowerShell commands have a very good, very intuitive naming convention of a verb followed by a noun (for example, `get-process` , `invoke-webrequest`), but I'm never quite sure where 'to' and 'from' go for the conversion commands.

To quickly look it up I can type:

`get-command *csv*` ... which returns:

```
$ get-command *csv*

CommandType     Name                    ModuleName
-----------     ----                    ----------
Alias           epcsv -> Export-Csv
Alias           ipcsv -> Import-Csv
Cmdlet          ConvertFrom-Csv         Microsoft.PowerShell.Utility
Cmdlet          ConvertTo-Csv           Microsoft.PowerShell.Utility
Cmdlet          Export-Csv              Microsoft.PowerShell.Utility
Cmdlet          Import-Csv              Microsoft.PowerShell.Utility
Application     ucsvc.exe
Application     vmicsvc.exe
```

Functions

Typically PowerShell coding is done in the form of *functions*[4]. What you do to code and write a function is this:

Create a function in a plain text .ps1 file[5]

```
gvim say-HelloWorld.ps1
```

...then source the function when they need it

```
$ . .\say-HelloWorld.ps1
```

...then run it

```
$ say-helloworld
Hello, World
```

Often people autoload their functions in their `$profile` or other startup script, as follows:

```
write-verbose "About to load functions"
foreach ($FUNC in $(dir $FUNCTION_DIR\*.ps1))
{
  write-verbose "Loading $FUNC.... "
  . $FUNC.FullName
}
```

Footnotes

[1] If you wanted the equivalent of `ls -ltr` you would use `gci | sort lastwritetime`. 'gci' is an alias for 'get-childitem', and I think, 'sort' is an alias for 'sort-object'.

[2] Another way of returning all of the properties of an object is to use 'select *'...so in this case you could type `get-process | select *`

[3] There is actually a built-in alias `man` which tranlates to `get-help`, so you can just type `man` if you're pining for Unix.

[4] See the following for more detail on writing functions rather than scripts: http://blogs.technet.com/b/heyscriptingguy/archive/2011/06/26/don-t-write-scripts-write-powershell-functions.aspx

[5] I'm using 'gvim' here, but notepad would work just as well. PowerShell has a free 'scripting environment' called *PowerShell ISE*, but you don't have to use it if you dont want to.

commands summary

alias (set aliases)

```
set-alias
```

More

alias (show aliases)

```
get-alias
```

More

apropos

```
get-help
```

More

basename

```
dir | select name
```

More

cal

No equivalent, but see the script at http://www.vistax64.com/powershell/17834-unix-cal-command.html

cd

```
cd
```

More

clear

```
clear-host
```

More

date

```
get-date
```

More

date -s

```
set-date
```

More

df -k

```
Get-WMIObject Win32_LogicalDisk | ft -a
```

More

diff

```
Compare-Object -ReferenceObject (Get-Content file1) -DifferenceObject (Get-Content file2)
```

dirname

```
dir | select directory
```

More

du

No equivalent, but see the link

echo

```
write-output
```

More

echo -n

```
write-host -nonewline
```

More

| egrep -i sql

```
 | where {[Regex]::Ismatch($\_.name.tolower(), "sql") }
```

More

egrep -i

```
select-string
```

More

egrep

```
select-string  -casesensitive
```

More

egrep -v

```
select-string -notmatch
```

More

env

```
Get-ChildItem Env: | fl
```

or

```
get-variable
```

More

errpt

```
get-eventlog
```

More

export PS1=”$ “

```
function prompt {"$ " }
```

More

find

```
dir  *whatever* -recurse
```

More

for (start, stop, step)

```
for ($i = 1; $i -le 5; $i++) {whatever}
```

More

head

```
gc file.txt | select-object -first 10
```

More

history

```
get-history
```

More

history | egrep -i ls

```
history | select commandline | where commandline -like '*ls*' | fl
```

More

hostname

```
hostname
```

More

if-then-else

```
if  ( condition ) { do-this } elseif { do-that } else {do-theother}
```

More

if [-f "$FileName"]

```
if (test-path $FileName)
```

More

kill

```
stop-process
```

More

less

```
more
```

More

locate

```
no equivalent but see link
```

More

ls

```
get-childitem OR gci OR dir OR ls
```

More

ls -a

```
ls -force
```

More

ls -ltr

```
dir c:\ | sort-object -property lastwritetime
```

More

lsusb

```
gwmi Win32_USBControllerDevice
```

More

mailx

```
send-mailmessage
```

More

man

```
get-help
```

More

more

```
more
```

More

mv

```
rename-item
```

More

pg

```
more
```

More

ps -ef

```
get-process
```

More

ps -ef | grep oracle

```
get-process oracle
```

More

pwd

```
get-location
```

More

read

```
read-host
```

More

rm

```
remove-item
```

More

script

```
start-transcript
```

More

sleep

```
start-sleep
```

More

sort

```
sort-object
```

More

sort -uniq

```
get-unique
```

More

tail

```
gc file.txt | select-object -last 10
```

More

tail -f

```
gc -tail 10 -wait file.txt
```

More

time

```
measure-command
```

More

touch - create an empty file

```
set-content -Path ./file.txt -Value $null
```

More

touch - update the modified date

```
set-itemproperty -path ./file.txt -name LastWriteTime -value $(get-date)
```

More

wc -l

```
gc ./file.txt | measure-object | select count
```

More

whoami

```
[Security.Principal.WindowsIdentity]::GetCurrent() | select name
```

More

whence or type

```
No direct equivalent, but see link
```

More

unalias

```
remove-item -path alias:aliasname
```

More

uname -m

```
Get-WmiObject -Class Win32_ComputerSystem | select manufacturer, model
```

More

uptime

```
get-wmiobject -class win32_operatingsystem | select LastBootUpTime`
```

More

\ (line continuation)

```
` (a backtick)
```

More

commands detail - a

alias (list all the aliases)

The Powershell equivalent of typing `alias` at the bash prompt is:

```
get-alias
```

alias (set an alias)

At it's simplest, the powershell equivalent of the unix 'alias' when it's used to set an alias is 'set-alias'

```
set-alias ss select-string
```

However, there's a slight wrinkle....

In unix, you can do this

```
alias bdump="cd /u01/app/oracle/admin/$ORACLE_SID/bdump/"
```

If you try doing this in Powershell, it doesn't work so well. If you do this:

```
set-alias cdtemp "cd c:\temp"
cdtemp
```

...then you get this error:

```
cdtemp : The term 'cd c:\temp' is not recognized as the name of a cmdlet, function, scrip
At line:1 char:1
+ cdtemp
+ ~~~~~~
    + CategoryInfo          : ObjectNotFound: (cd c:\temp:String) [], CommandNotFoundExce
    + FullyQualifiedErrorId : CommandNotFoundException
```

A way around this is to create a function instead:

```
remove-item -path alias:cdtemp
function cdtemp {cd c:\temp}
```

You can then create an alias for the function:

```
set-alias cdt cdtemp
```

apropos

`apropos` is one of my favourite bash commands, not so much for what it does...but because I like the word 'apropos'.

I'm not sure it exists on all flavours of *nix, but in bash `apropos` returns a list of all the man pages which have something to do with what you're searching for. If apropos isn't implemented on your system you can use `man -k` instead.

Anyway on bash, if you type:

```
apropos process
```

...then you get:

```
AF_LOCAL [unix]      (7)  - Sockets for local interprocess communication
AF_UNIX [unix]       (7)  - Sockets for local interprocess communication
Apache2::Process     (3pm)  - Perl API for Apache process record
BSD::Resource        (3pm)  - BSD process resource limit and priority functions
CPU_CLR [sched_setaffinity] (2)  - set and get a process's CPU affinity mask
CPU_ISSET [sched_setaffinity] (2)  - set and get a process's CPU affinity mask
CPU_SET [sched_setaffinity] (2)  - set and get a process's CPU affinity mask
CPU_ZERO [sched_setaffinity] (2)  - set and get a process's CPU affinity mask
GConf2              (rpm) - A process-transparent configuration system
```

The Powershell equivalent of `apropos` or `man -k` is simply `get-help`

```
get-help process

Name                     Category  Module     Synopsis
----                     --------  ------     --------
get-dbprocesses          Function             Get processes for a particul...
show-dbprocesses         Function             Show processes for a particu...
Debug-Process            Cmdlet    Microso... Debugs one or more processes...
Get-Process              Cmdlet    Microso... Gets the processes that are ...
```

This is quite a nice feature of PowerShell compared to Bash. If `get-help` in Powershell shell scores a 'direct hit' (i.e. you type something like `get-help debug-process`) it will show you the help for that particular function. If you type something more vague, it will show you a list of all the help pages you might be interested in.

By contrast if you typed `man process` at the Bash prompt, you'd just get

```
No manual entry for process
```

commands detail - b

basename

A rough PowerShell equivalent for the unix *basename* is:

```
dir <whatever> | select name
```

This depends on the file actually existing, whereas *basename* doesn't care.

A more precise (but perhaps less concise) alternative[1] is:

`[System.IO.Path]::GetFileName('c:\temp\double_winners.txt')`

Notes [1] I found `[System.IO.Path]::GetFileName` after reading Power Tips of the Day - Useful Path Manipulations Shortcuts, which has some other useful commands

commands detail - c

cal

There's no one-liner equivalent for the Linux `cal` , but there's a useful script, with much of the `cal` functionality here :

http://www.vistax64.com/powershell/17834-unix-cal-command.html

cd

The PowerShell equivalent of `cd` is:

```
Set-Location
```

...although there is a builtin PowerShell alias `cd` which points at `set-location`

cd ~

`cd ~` moves you to your home folder in both unix and Powershell.

clear

The unix `clear` command clears your screen. The Powershell equivalent to the unix `clear` is

```
clear-host
```

PowerShell also has built-in alias `clear` for `clear-host` .

However, it's possibly worth noting that the behaviour of the two commands is slightly different between the two environments.

In my Linux environment, running putty, `clear` gives you a blank screen by effectively scrolling everything up, which means you can scroll it all back down.

The Powershell `Clear-host` on the other hand seems to wipe the previous output (actually in the same way that cmd's `cls` command does....). This *could* be quite a significant difference, depending on what you want to clear and why!

cp

The Posh version of cp is

```
copy-item
```

The following are built-in aliases for copy-item:

```
cp
copy
```

cp -R

To recursively copy:

```
copy -recurse
```

commands detail - d

date

The Powershell equivalent of the Unix `date` is

```
get-date
```

The Powershell equivalent of the Unix `date -s` is

```
set-date
```

I was anticipating doing a fairly tedious exercise of going through all the Unix date formats and then working out the Powershell equivalent, but discovered the Powershell Team has effectively done all this for me. There is a Powershell option `-UFormat` which stands for 'unix format'.

So the Powershell:

```
date -Uformat '%D'
09/08/14
```

is the same as the *nix

```
date +'%D'
09/08/14
```

This is handy...but I have found the odd difference. I tried this for a demo:

Unix:

```
date +'Today is %A the %d of %B, the %V week of the year %Y. My timezone is %Z, and here
Today is Monday the 08 of September, the 37 week of the year 2014. My timezone is BST, an
```

Powershell:

```
get-date -Uformat 'Today is %A the %d of %B, the %V week of the year %Y. My timezone is %
Today is Monday the 08 of September, the 36 week of the year 2014. My timezone is +01, an
```

I presume the discrepancy in the week of the year is to do with when the week turns - as you can see I ran the command on a Monday. Some systems have the turn of the week being Monday, others have it on Sunday.

I don't know why `%Z` outputs different things....and I can't help feeling I'm being churlish pointing this out. The -UFormat option is a really *nice* thing to have.

df -k

A quick and dirty Powershell equivalent to 'df -k' is

```
Get-WMIObject Win32_LogicalDisk -filter "DriveType=3" | ft
```

A slightly prettier version is this function:

```
function get-serversize { Param( [String] $ComputerName)

Get-WMIObject Win32_LogicalDisk -filter "DriveType=3" -computer $ComputerName |
   Select SystemName, DeviceID, VolumeName,
          @{Name="size (GB)";Expression={"{0:N1}" -f($_.size/1gb)}},
          @{Name="freespace (GB)";Expression={"{0:N1}" -f($_.freespace/1gb)}}
}

function ss { Param( [String] $ComputerName)
    get-serversize $ComputerName | ft
}
```

....then you can just do:

```
$ ss my_server
```

....and get

```
SystemName   DeviceID    VolumeName     size(GB)          freespace(GB)
----------   --------    ----------     --------          -------------
my_server    C:          OS             30.0                        7.8
my_server    D:          App            250.0                       9.3
my_server    E:                         40.0                        5.0
```

dirname

A good PowerShell equivalent to the unix `dirname` is

```
gi c:\double_winners\chelsea.doc | select directory
```

However, this isn't a *direct* equivalent. Here, I'm telling Powershell to look at an actual file and then return that file's directory. The file has to exist. The unix 'dirname' doesn't care whether the file you specify exists or not. If you type in `dirname /tmp/double_winners/chelsea.doc` on *any* Unix server it will return `/tmp/double_winners`, I think. `dirname` is essentially a string-manipulation command.

A more precise Powershell equivalent to the unix 'dirname' is this

```
[System.IO.Path]::GetDirectoryName('c:\double_winners\chelsea.doc')
```

....but it's not as easy to type, and 9 times out of 10 I do want to get the folder for an existing file rather than an imaginary one.

du

While I think there *are* implementations of `du` in PowerShell, personally my recommendation would be to download Mark Russinovich's 'du' tool, which is here:

Windows Sysinternals - Disk Usage

This is part of the Microsoft's 'sysinternals' suite.

commands detail - e

echo

`echo` is an alias in PowerShell. As you would expect it's an alias for the closest equivalent to the Linux `echo` :

- `write-output`

You use it as follows:

```
write-output "Blue is the colour"
```

As well as write-output there are a couple of options for use in Powershell scripts and functions:

- `write-debug`
- `write-verbose`

Whether these produce any output is controlled by commandline or environment flags.

echo -n

In bash, `echo -n` echoes back the string without printing a newline, so if you do this:

```
$ echo -n Blue is the colour
```

you get:

```
Blue is the colour$
```

....with your cursor ending up on the same line as the output, just after the dollar prompt

Powershell has an exact equivalent of 'echo -n'. If you type:

```
PS C:\Users\matt> write-host -nonewline "Blue is the colour"
```

....then you get this:

```
PS C:\Users\matt> write-host -nonewline "Blue is the colour"
Blue is the colourPS C:\Users\matt>
```

Note that `-nonewline` doesn't 'work' if you're in the ISE.

egrep

The best PowerShell equivalent to `egrep` or `grep` is `select-string` :

```
select-string stamford blue_flag.txt
```

A nice feature of `select-string` which *isn't* available in `grep` is the `-context` option. The -context switch allows you to see a specified number of lines either side of the matching one. I think this is similar to `SEARCH /WINDOW` option in DCL.

egrep -i

Powershell is case-insensitive by default, so:

```
select-string stamford blue_flag.txt
```

...would return:

```
blue_flag.txt:3:From Stamford Bridge to Wembley
```

If you want to do a case sensitive search, then you can use:

```
select-string  -casesensitive stamford blue_flag.txt
```

egrep -v

The Powershell equivalent to the `-v` option would be `-notmatch`

```
select-string -notmatch stamford blue_flag.txt
```

egrep 'this|that'

To search for more than one string within a file in bash, you use the syntax:

```
egrep 'blue|stamford' blue_flag.txt
```

This will return lines which contain either 'blue' or 'stamford'.

The PowerShell equivalent is to seperate the two strings with a comma, so:

```
$ select-string  stamford,blue blue_flag.txt
```

...returns:

```
blue_flag.txt:2:We'll keep the blue flag flying high
blue_flag.txt:3:From Stamford Bridge to Wembley
blue_flag.txt:4:We'll keep the blue flag flying high
```

| egrep -i sql

This is an interesting one, in that it points up a conceptual difference between PowerShell and Bash.

In bash, if you want to pipe into a grep, you would do this:

```
ps -ef | egrep sql
```

This would show you all the processes which include the string 'sql' somewhere in the line returned by `ps`. The egrep is searching across the whole line. If the username is 'mr_sql' then a line would be returned, and if the process is 'sqlplus' than a line would also be returned.

To do something similar in PowerShell you would do something more specific

```
get-process | where processname -like '*sql*'
```

So the string 'sql' has to match the contents of the property `processname`. As it happens, get-process by default only returns one text field, so in this case it's relatively academic, but hopefully it illustrates the point.

env

The Linux 'env' shows all the environment variables.

In PowerShell there are two set of environment variables:

- windows-level variables and
- Powershell-level variable

Windows-level variables are given by:

```
Get-ChildItem Env: | fl
```

PowerShell-level variables are given by:

```
get-variable
```

errpt

I think errpt is possibly just an AIX thing (the linux equivalent is, I think, looking at `/var/log/message`). It shows system error and log messages.

The PowerShell equivalent would be to look at the Windows eventlog, as follows

```
get-eventlog -computername bigserver -logname application -newest 15
```

The lognames that I typically look at are 'system', 'application' or 'security'.

export PS1="$ "

In bash the following changes the prompt when you are at the command line

```
export PS1="$ "
```

The Powershell equivalent to this is:

```
function prompt {
 "$ "
 }
```

I found this on Richard Siddaway's Blog

commands detail - f

find

The bash `find` command has loads of functionality - I could possibly devote many pages to Powershell equivalents of the various options, but at it's simplest the bash `find` does this:

```
find . -name '*BB.txt'
./Archive/Script_WO7171BB.txt
./Archive/Script_WO8541BB.txt
./Archive/Script_WO8645_BB.txt
./Archive/WO8559B/Script_WO8559_Master_ScriptBB.txt
./Archive/WO8559B/WO8559_finalBB.txt
./Archive/WO8559B/WO8559_part1BB.txt
./Archive/WO8559B/WO8559_part2BB.txt
```

The simplest Powershell equivalent of the bash `find` is simply to stick a `-recurse` on the end of a `dir` command

```
PS x:\> dir  *BB.txt -recurse

    Directory: x:\Archive\WO8559B

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-----        28/02/2012     17:15        608 Script_WO8559_Master_ScriptBB.txt
-----        28/02/2012     17:17         44 WO8559_finalBB.txt
-----        28/02/2012     17:17      14567 WO8559_part1BB.txt
-----        28/02/2012     17:15       1961 WO8559_part2BB.txt

    Directory: x:\Archive

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-----        15/06/2011     08:56       2972 Script_WO7171BB.txt
-----        14/02/2012     16:39       3662 Script_WO8541BB.txt
-----        27/02/2012     15:22       3839 Script_WO8645_BB.txt
```

If you want Powersehll to give you output that looks more like the Unix find then you can pipe into `| select fullname`

```
PS x:\> dir  *BB.txt -recurse | select fullname

FullName
--------
x:\Archive\WO8559B\Script_WO8559_Master_ScriptBB.txt
x:\Archive\WO8559B\WO8559_finalBB.txt
x:\Archive\WO8559B\WO8559_part1BB.txt
x:\Archive\WO8559B\WO8559_part2BB.txt
x:\Archive\Script_WO7171BB.txt
x:\Archive\Script_WO8541BB.txt
x:\Archive\Script_WO8645_BB.txt
```

for

for loop - start, stop, step

The equivalent of this bash:

```
for (( i = 1 ; i <= 5 ; i++ ))
do
  echo "Hello, world $i"
done

Hello, world 1
Hello, world 2
Hello, world 3
Hello, world 4
Hello, world 5
```

...is

```
for ($i = 1; $i -le 5; $i++)
{
  write-output "Hello, world $i"
}

Hello, world 1
Hello, world 2
Hello, world 3
Hello, world 4
Hello, world 5
```

for loop - foreach item in a list

For the Bash

```
for I in Chelsea Arsenal Spuds
do
  echo $I
done
```

the equivalent Powershell is:

```
foreach ($Team in ("Chelsea", "Arsenal", "Spuds")) {write-output $Team}
```

for loop - for each word in a string

For the bash:

```
london="Chelsea Arsenal Spurs"
for team in $london; do   echo "$team"; done
```

...the equivalent Powershell is:

```
$London = "Chelsea Arsenal Spuds"
foreach ($Team in ($London.split())) {write-output $Team}
```

for loops - for lines in a file

Bash:

```
for team in $(egrep -v mill london.txt)
> do
>   echo $team
> done
```

Posh:

```
select-string -notmatch millwall london.txt | select line | foreach {write-output $_}
```

or:

```
foreach ($team in (select-string -notmatch millwall london.txt | select line)) {$team}
```

for loop - for each file in a folder</h4>

Bash:

```
for LocalFile in *
do
  echo $LocalFile
done
```

Posh:

```
foreach ($LocalFile in $(gci)) {write-output $LocalFile.Name}
```

commands detail - g

Not got any commands beginning with 'g' yet.

commands detail - h

head

The PowerShell equivalent of the *nix `head` is:

```
gc file.txt | select-object -first 10
```

history

The Powershell equivalent of `history` is:

```
get-history
```

There is a built in alias `history`

It's worth noting that history doesn't persist across PowerShell sessions, although if you search online there are a couple of published techniques for making it persistent.

It's also perhaps worth noting that Powershell gives you a couple of extra bits of information, if you want them:

```
get-history | gm -MemberType Property


   TypeName: Microsoft.PowerShell.Commands.HistoryInfo

Name               MemberType Definition
----               ---------- ----------
CommandLine        Property   string CommandLine {get;}
EndExecutionTime   Property   datetime EndExecutionTime {get;}
ExecutionStatus    Property   System.Management.Automation.Runspaces.PipelineState Execut
Id                 Property   long Id {get;}
StartExecutionTime Property   datetime StartExecutionTime {get;}
```

history | egrep -i ls

There is no direct equivalent of the shell functionality you get with `set -o vi` sadly. You can up- and down- arrow by default, but if you want to search through your history then you need to do something like this

```
history | select commandline | where-object {$_.commandline -like '*ls*'} | fl
```

hostname

There is a windows `hostname` which does much the same thing as the Unix `hostname`, but it's not Powershell. It's a standard-ish Windows executable that on my machine lives in c:\windows\system32

Details are here: [Microsoft Windows XP - Hostname]

You can get the server name through PowerShell like this:

```
get-wmiobject -class win32_operatingsystem | select __SERVER
```

commands detail - i

if-then-else

The bash `if-then-elif-else` as per:

```
HOUR_OF_DAY=$(date +'%H')

if  [ $HOUR_OF_DAY -lt 6 ]
then
  echo  "Still nightime"
elif [ $HOUR_OF_DAY -lt 12 ]
then
  echo "Morning has broken"
elif [ $HOUR_OF_DAY -lt 18 ]
then
  echo "After noon"
else
  echo "Nightime again"
fi
```

...could be rendered in PowerShell as:

```
[int]$HourOfDay = $(get-date -UFormat '%H')

if  ( $HourOfDay -lt 6 )
{
  write-output  "Still nightime"
}
elseif ( $HourOfDay -lt 12 )
{
  write-output "Morning has broken"
}
elseif ( $HourOfDay -lt 18 )
{
  write-output "After noon"
}
else
{
  write-output "Nightime again"
}
```

if [-f "$FileName"]

Testing for the existence of a file in bash is done as follows

```
export FileName=~/.matt
if [ -f "$FileName" ]
then
  echo "$FileName found."
else
  echo "$FileName not found."
fi
```

In PowerShell this could be[1]

```
$FileName = "c:\powershell\.matt.ps1x"
if (test-path $FileName)
  {echo "$FileName found"}
else
  {echo "$FileName not found"}
```

Footnotes

[1] The way I've rendered the PowerShell here isn't great, but I've left it like that because for a couple of reasons. First, it shows the similarity between PowerShell and Bash, which I think is encouraging for anyone reading this e-book. Second it allows me make a brief point about using aliases.

`echo` is handy. It's short, and it looks like it does the same thing as `echo` in Unix, MS-DOS and probably a few other languages besides. It pretty much does...but does `echo` alias `write-output` which allows you to pipe to other PowerShell commands, or does it alias to `write-host`, which doesn't?

I've been using PowerShell for a few years now but I didn't know. I had to look it up. This is extra hassle if you're reading a script, which is one of the reasons that it's usually seen as being better practice in scripts to be explicit by using the full command rather than the alias.

Also, in PowerShell scripts rather than this:

```
if (test-path $FileName)
  {write-host "$FileName found"}
```

...it would typically be seen as better to format using one of these two alternatives:

```
if (test-path $FileName) {
  write-host "$FileName found"
}
```

or:

```
if (test-path $FileName)
{
  write-host "$FileName found"
}
```

commands detail - j

None as yet

commands detail - k

kill

The equivalent of bash's `kill` is:

```
stop-process
```

A typical usage in Powershell might be:

```
# find the process
get-process | select id, ProcessName | where {$_.processname -like 'iex*'}

# kill the process
stop-process 5240
```

There is a built in alias `kill` which translates to stop-process

```
get-alias k*

CommandType     Name
-----------     ----
Alias           kill -> Stop-Process
```

commands detail - l

locate

There isn't a builtin PowerShell version of locate, but Chrissy LeMaire's (website) has written an Invoke-Locate script 'in the spirit of (Linux/Unix) GNU findutils' locate'. It works really well

ls

The PowerShell equivalent of the Unix *ls* is:

```
Get-ChildItem
```

... for which there are aliases *dir*, *ls* and *gci*

ls -a

In linux, *ls -a* displays hidden files as well as 'normal' files.

So *ls* gives:

```
$ ls
README.md
```

but *ls -a* gives

```
$ ls -a
.   ..   .function-prompt.ps1.swp   .git   README.md
```

The Powershell equivalent of *ls -a* is *get-childitem -force*. Here, I've used the alias *ls*

```
$ ls


    Directory: C:\Users\matt\Documents\WindowsPowerShell\functions


Mode                LastWriteTime     Length Name
----                -------------     ------ ----
-a---        04/06/2015     13:20       1422 README.md




$ ls -force


    Directory: C:\Users\matt\Documents\WindowsPowerShell\functions


Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d--h-        04/06/2015     13:20            .git
-a-h-        20/05/2015     17:33      12288 .function-prompt.ps1.swp
-a---        04/06/2015     13:20       1422 README.md
```

ls -ltr

The Powershell equivalent of the unix *ls -ltr* (or the DOS *dir /OD*), which lists files last update order.

```
dir c:\folder | sort-object -property lastwritetime
```

lsusb

The unix command *lsusb* shows USB devices. The PowerShell equivalent is:

```
gwmi Win32_USBControllerDevice
```

gwmi is an alias for *get-wmiobject*

commands detail - m

mailx

To send an email from the PowerShell command line, this worked for me:

```
$PSEmailServer = "exchange_server.domain.co.uk"
send-mailmessage -to eden.hazard@gmail.com -from matt@here.co.uk -subject "Hello"
```

man

The Powershell equivalent of `man` is:

```
get-help
```

`get-help` has the following built-in aliases:

- `help`
- `man`

There are a couple of things to note about `get-help`.

There are two much-used options: `-full` and `-examples`. They both do exactly what you'd expect, I think. To give some idea of scale, on my laptop `get-help get-process` currently returns just over a screenful of information, wherease `get-help -get-process -full` returns 9 screenfuls.

The help text can be brought up-to-date by running update-help from the command line.

You can easily write your own help text for your own functions, by using a feature called *comment-based help*.

man -k

In *nix `man -k` allows you to search through all the man pages for mentions of a particular keyword. It returns a list of the man pages which are relevant to the word you've searched for. On some systems, it's aliased to `apropos`. Anyway, `man -k disk` would perhaps return

lines for, say, `du`, `df` and `lsvol` (at the time of typing I don't have a Linux install to hand, so I'm guessing here.)

There's no seperate command for this in PowerShell, because the `get-help` command does this by default if it doesn't find a direct match.

So, if you type `get-help get-process` you would get this:

```
NAME
    Get-Process

SYNOPSIS
    Gets the processes that are running on the local computer or a remote computer.


SYNTAX
    Get-Process [[-Name] <String[]>] [-ComputerName <String[]>] [-FileVersionInfo] [-Modu

    Get-Process [-ComputerName <String[]>] [-FileVersionInfo] [-Module] -InputObject <Pro

etc....
```

...whereas if you typed `get-help process` you would get a list of help topics related to 'process'[1]:

```
Name          Category Synopsis
----          -------- --------
Debug-Process Cmdlet   Debugs one or more processes running on the local computer.
Get-Process   Cmdlet   Gets the processes that are running on the local computer or a rem
Start-Process Cmdlet   Starts one or more processes on the local computer.
Stop-Process  Cmdlet   Stops one or more running processes.
Wait-Process  Cmdlet   Waits for the processes to be stopped before accepting more input.
```

more

Powershell incorporates a `more` command which broadly works in the console similarly to the unix `more`.

The Powershell `more` is a wrapper for `more.com` [2], which is an old Microsoft implementation of `more`.

`more` doesn't work in the ISE, but you can however easily scroll back through output by pressing 'Ctrl' and 'Up-arrow' at the same time. This then allows you to use all the arrow keys (as well as Ctrl-c and Ctrl-V to cut and paste) to navigate around the output from

previous commands.

mv

The PowerShell equivalent of `mv` is:

```
Rename-Item
```

Footnotes

---

[1] To be honest, I actually did `get-help process | select name, category, synopsis | ft -a` to tidy up the output for the e-book.

[2] I found that in my current PowerShell installs, there wasn't much information on `more`. The `get-help` command returned the barest of details.

To see what the command actually does I ran:

```
get-command more | select definition | format-list
```

commands detail - n

Nothing yet

commands detail - o

Nothing for commands beginning with 'o' yet.

commands detail - p

ps

The PowerShell equivalent of the `ps` command is:

```
get-process
```

You can use `get-process` to get information about other computers:

```
get-process -ComputerName bigserver gvim*
```

You can use `select` and `where` to 'slice and dice' the information.

```
get-process  |
  where {$_.PeakWorkingSet -gt 1Mb } | select ProcessName,PeakWorkingSet
```

As with `ps`, the `get-process` command has many options. This section of the e-book will be expanded over the next few months but, to start with, these are some of the `ps` examples from the Linux `man` page.

ps -ef (see every process on the system)

By default `get-process` shows all of the processes on the current PC or server

ps (show just current process)

If you wanted to just see details of your process you could do this:

```
get-process -pid $PID
```

ps -ejH (print a process tree)

There is no PowerShell equivalent to the Unix `ps -eJH`, because as I understand it Windows processes aren't part of a process tree.

ps -eLf (get info about threads)

I *think* this shows information about the processes threads:

```
get-process -pid $pid | select -expand threads
```

ps -U (show particular user)

```
get-process -IncludeUserName | ? Username -eq "Ronnie\Matt"
```

ps -ef | grep firefox

```
get-process firefox
```

pwd

To show your current location in Powershell:

```
Get-Location
```

...or there are aliases `gl` and `pwd` .

There is also a built-in variable

```
$pwd
```

commands detail - q

Write here...

commands detail - r

read -p

In *nix:

```
read -p "Which is the only London club to win the Champions League? " team
echo $team
```

In Powershell:

```
$team = read-host "Which is the only London club to win the Champions League? "
Which is the only London club to win the Champions League? : Chelsea
$team
Chelsea
```

To *not* echo the input to screen, you would do

```
$SecretString = read-host "Whats your secret? "-assecurestring
```

This echoes out an asterisk for each character input

rm

```
Remove-Item
```

commands detail - s

script

```
start-transcript
```

sleep

```
start-sleep -seconds 5

or

start-sleep -milliseconds 250
```

or just:

```
sleep 3
```

...will sleep for 3 seconds

sort

```
get-process | sort-object VirtualMemorySize
```

sort -u

The closest PowerShell equivalent to the unix `sort -u` is `get-unique`

```
gc c:\temp\2000.txt | sort | gu
```

Note: this only works as far I can see if you sort it first

Note 2: get-unique IS case sensitive

sql

This isn't really a Powershell equivalent of a unix command, but in case it's useful, to call Sqlserver's implementation of the sql command line from Powershell you can use `invoke-sqlcmd`

```
Invoke-Sqlcmd -ServerInstance -query "Select blah" -database _catalog
```

You need to have the sql module loaded for this to work, or be running the Powershell console from within SSMS

commands detail - t

tail

```
gc file.txt | select-object -last 10
```

tail -f

```
gc -tail 10 -wait c:\windows\windowsupdate.log
```

tee

The Powershell equivalent of the unix *tee* is *tee-object*....which, by default is aliased to *tee*

So you can do this:

```
get-process | tee c:\temp\test_tee.txt
```

...to both get a list of processes on your screen and get that output saved into the file in c:\temp

time

The Powershell equivalent of the bash shell 'time' is 'measure-command'.

So, in bash you would do this:

```
time egrep ORA- *log
```

....and get all the egrep output, then

```
real    0m4.649s
user    0m0.030s
sys     0m0.112s
```

In Powershell, you would do this

```
measure-command {select-string ORA- *.sql}
```

...and get...

```
Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 105
Ticks             : 1057357
TotalDays         : 1.22379282407407E-06
TotalHours        : 2.93710277777778E-05
TotalMinutes      : 0.00176226166666667
TotalSeconds      : 0.1057357
TotalMilliseconds : 105.7357
```

...you don't get the 'user CPU' time and 'system CPU' time, but you do get the added bonus of seeing how long the command took rendered as a fraction of a day!

touch - create an empty file

```
set-content -Path c:\temp\new_empty_file.dat -Value $null
```

I found the set-content command at Super User, the contributor being user techie007

touch - update the modified date

```
set-itemproperty -path c:\temp\new_empty_file.dat -name LastWriteTime -value $(get-date)
```

I got this from a comment by Manung Han on the Lab49 Blog. Doug Finke shares touch function in a later comment on the same post that fully implements the linux command.

commands detail - u

unalias

```
remove-item -path alias:cdtemp
```

uname

uname -s

`uname -s` in Unix, according to the man page, gives the 'kernel-version' of the OS. This is the 'top-level version' of the Unix that you're on. Typical values are 'Linux', or 'AIX' or 'HP-UX'. So, on my laptop, typing `uname -s` gives:

```
Linux
```

I've only used this when writing a Unix script which have to do slightly different things on different flavours of unix.

Obviously, there's only one manufacturer for Windows software - Microsoft. So there's no direct equivalent to `uname -s`. The closest equivalent on Powershell would I think be:

get-wmiobject -class win32_operatingsystem | select caption

This returns:

```
caption
-------
Microsoft Windows 7 Professional
```

or

```
Microsoft Windows 8.1 Pro
```

or

```
Microsoft(R) Windows(R) Server 2003, Standard Edition
```

or

```
Microsoft Windows Server 2008 R2 Enterprise
```

or

```
Microsoft Windows Server 2012 Standard
```

uname -n

According to the Linux help, uname -n does this:

```
       -n, --nodename
              print the network node hostname
```

So, typing uname -n gives

```
$ uname -n
nancy.one2one.co.uk
```

I haven't found a neat equivalent for this in Powershell, but this works:

```
get-wmiobject -class win32_computersystem  | select dnshostname, domain
```

The output is:

```
dnshostname                                                  domain
-----------                                                  ------
nancy                                                        one2one.co.uk
```

uname -r

uname -r gives the kernel release in Unix. The output varies depending on the flavour of Unix - Wikipedia has a good list of examples

On my system uname -r gives:

```
2.6.32-200.20.1.el5uek:
```

The best Powershell equivalent would seem to be:

```
get-wmiobject -class win32_operatingsystem | select version
```

...which gives:

```
6.1.7601
```

The 7601 is Microsoft's build number.

uname -v

uname -v typically gives the date of the unix build. As far a I can think, there isn't a Powershell equivalent

uname -m

To be honest, I'm not entirely sure what uname -m shows us on Unix. The wikipedia page for uname shows various outputs none of which are hugely useful.

Running uname -m on my server gives:

```
x86_64
```

Is this a PowerShell equivalent?

```
$ get-ciminstance -class cim_computersystem | select SystemType
SystemType
----------
x64-based PC
```

uptime

On most, but from memory possibly not all, flavours of *nix 'uptime' tells you how long the server has been up and running

```
$ uptime
 15:54:24 up 9 days,  5:43,  2 users,  load average: 0.10, 0.09, 0.07
```

A rough Powershell equivalent to show how long the server (or PC) has been running is:

```
get-wmiobject -class win32_operatingsystem  | select LastBootUpTime
```

....of course you can also do

```
get-wmiobject -class win32_operatingsystem -ComputerName some_other_server |
    select LastBootUpTime
```

...to get the bootup time for a remote server, or PC.

commands detail - v

No commands beginning with 'v' so far.

commands detail - w

wc -l

```
gc c:\temp\file.txt | measure-object | select count
```

to show the number of *non-blank* lines:

```
gc c:\temp\file.txt | measure-object -line
```

whoami

This shows user that you are logged on as:

```
[Security.Principal.WindowsIdentity]::GetCurrent() | select name
```

whence or type

There isn't a single equivalent to the unix `whence` command, but there are a couple of things worth mentioning.

This shows the sort of thing (exe, bat, alias, function) that you're looking at:

```
get-command whoami

CommandType     Name                                                ModuleName
-----------     ----                                                ----------
Application     whoami.exe
```

....and if what you're looking for is a file in your path, then this will find it

```
foreach ($FOLDER in $ENV:PATH.split(";") ) { dir $FOLDER\whoami.exe -ea Si | select fulln

FullName
--------
C:\Windows\system32\whoami.exe
```

This splits the path into its constituent folders, then does a *dir* to see if the file (in this case I'm looking for *whoami.exe*) exists in each.

commands detail - x

Write here...

commands detail - y

Write here...

commands detail - z

Write here...

commands detail - non-alphabetical

Write here...

Todo

While the first version of this e-book is being written this list will be largely mechanical stuff which needs to be done to get the e-book into a suitable state. Subsequently it will be more a list of unix stuff for which I/we stilll need to find or document a PowerShell equivalent.

for version '1.0'

- more or the resources, perhaps
- check each page in pdf or word

for future versions

- look at http://blogs.technet.com/b/josebda/archive/2015/04/18/windows-powershell-equivalents-for-common-networking-commands-ipconfig-ping-nslookup.aspx

test conditions (not entirely sure that's the right unix terminology) - built test conditions like if file exists and is not a directory, if variable exists and is not null

pushd/popd

shutdown -r - restart-computer

more/less - remember it doesn't work in ISE

find - the various options. -newer, -exec

uname uname options

crontab -l cp -r ls -R .profile

bg cp cut

env eval file find free (memory) fuser filename head

tee

/var/log/message write & (run in background) PS1 (line contunuation prompt) declare -F type Parameter passing cut -f 3 for (p127) while (p139) until case select p113, p136 String comparisons p118 File attribute operations p122 fileinfo Number comparisons p126 IFS

(internal field separator) p127 PS3 getopts p145 let p145 arrays p160 here -documents p165 debugging stuff p221 -n (syntax check) -v -x

For the section on ' | egrep -i' i.e. how to search for something within the pipeline, I've currently got instructions on how to use -like against a particular property. It would be good to have an alternative which did allow you to search across the whole output. Not very useful typically, but it would be nice to cover it off

export (variables)

my search history function

Mark L's comments

- would expect to see stuff like 'if-then-else' in the introductory pages
- would be worth looking at the man pages for bash itself (and perhaps for cmd)
- cover re-direction
- 'special variables' - $HOME, $PROFILE

More detail on invoke-locate ?

Cover Get-Item as well as Get-ChildItem for ls

Convert from gwmi to get-ciminstance

More on mailx/send-mailmessage

Think about whether to expand any and all aliases to the full command name

More on mv?

Fill out detail on the ps process tree option. All unix processes being descendants of root, windows processes not necessarily being descendants of anything

More on more :). More isn't an alias for out-host -paging

ps options - starting with those in the cygwin or bash help

ps -0 (get security info) ps -eo euser,ruser,suser,fuser,f,comm,label ps axZ ps -eM

....have been looking at the cim but not got anything much yet. http://powershell.com/cs/blogs/tips/archive/2009/12/17/get-process-owners.aspx has wmi

ps -o

```
To see every process with a user-defined format:
   ps -eo pid,tid,class,rtprio,ni,pri,psr,pcpu,stat,wchan:14,comm
   ps axo stat,euid,ruid,tty,tpgid,sess,pgrp,ppid,pid,pcpu,comm
   ps -eopid,tt,user,fname,tmout,f,wchan
```

ps -C

```
Print only the process IDs of syslogd:
   ps -C syslogd -o pid=
```

ps -p

```
Print only the name of PID 42:
   ps -p 42 -o comm=
```

rm options

rmdir

sort and sort -uniq - more detail on each

uptime - restore the 'sos' function etc....but work out what the approved verb would be for 'show'

who am i - as opposed to whoami. I think it shows the user you originally logged on as

the non-alphabetical stuff: \ and backtick

$ env | sort _=/bin/env CVS_RSH=ssh G_BROKEN_FILENAMES=1 HISTSIZE=1000 HOME=/home/matt HOSTNAME=whatever.co.uk INPUTRC=/etc/inputrc LANG=en_GB LESSOPEN=|/usr/bin/lesspipe.sh %s LOGNAME=matt LS_COLORS=no=00:fi=00:di=00;34:ln=00;36:pi=40;33:so=00;35:bd=40;33;01:cd=40;33;01:or=01;05;37;41:mi=01;05;37;41:ex=00;32:.*cmd=00;32:*.exe=00;32:.*com=00;32:*.btm=00;32:.*bat=00;32:*.sh=00;32:.*csh=00;32:*.tar=00;31:.*tgz=00;31:*.arj=00;31:.*taz=00;31:*.lzh=00;31:.*zip=00;31:*.z=00;31:.*Z=00;31:*.gz=00;31:.*bz2=00;31:*.bz=00;31:.*tz=00;31:*.rpm=00;31:.*cpio=00;31:*.jpg=00;35:.*gif=00;35:*.bmp=00;35:.*xbm=00;35:*.xpm=00;35:.*png=00;35:*.tif=00;35: MAIL=/var/spool/mail/matt PATH=/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/home/matt/bin PS1=$ PWD=/home/matt SHELL=/bin/bash SHLVL=1 SSH_TTY=/dev/pts/1 TERM=xterm